PMON 2000

Command Reference Manual

R 3.x

Table of content

about

The **about** command displays information about PMON2000.

Command Format

```
about
```

The **about** command has no parameters.

Functional description

The **about** command display information and short history of PMON2000.

Environment

No environment variables affect this command.

See Also

The h command for displaying command information.

b

The **b** command set and display breakpoints.

Command Format

```
b
b addr...
b addr -s command
```

where:

addr	address where breakpoint should be inserted
- s *string*	executes the command string ***string*** when the breakpoint is reached. If no command is specified the command string in the **brkcmd** variable value will be executed. See below.

Functional description

The **b** command inserts a breakpoint at the specified address or addresses. Multiple address expressions may be specified at the same time. An address must comply with any alignment restrictions wich the target processor architecture has.

A breakpoint number is automatically assigned to each breakpoint when set. If multiple addresses are specified each address will have its own breakpoint number. When a breakpoint number is allocated the first unused number will be used.

PMON2000 reports a new breakpoints number immediately after the breakpoint is set (see the examples at the end of this subsection for an illustration of this). The assigned number can be used in the db command to specify the breakpoint to delete.

Each breakpoint can have a command string which is executed when the breakpoint is reached. The value of the string is set using the **-s *string*** parameter. For example, the string command:

```
PMON> b print -s "l @cpc 1;d -s @a0"
```

will print the instruction at the function print and then display the string pointed by argument register a0.

When a breakpoint is reached, which was set without using the **-s string** parameter, the commans specified in the environment variable brkcmd is executed. The default setting for brkcmd is:

```
brkcmd = "l @cpc 1"
```

This command "**l @cpc 1**", specifies that when the breakpoint occurs, PMON2000 will disassemble one instruction starting at the address in the current program counter register.

<

The breakpoint command variable **brkcmd** can be changed with the set command. For example, additional PMON2000 commands can be added in the brkcmd variable. Multiple commands must be separated using a semicolon. For example, the following command list will display the instruction at the breakpoint and then display all the register values.

```
PMON> set brkcmd "l @epc 1;r *"
```

By default, all breakpoints are removed when the load, or boot command is executed. See the section on each command in this document for details on how to override automatic breakpoint clearing after a download operation.

Some examples illustrating the use of the **b** command follow.

```
PMON> b a002000c          Set a break @ 0xa00200c
Bpt 1 = a002000c

PMON> b                   List all breakpoints
Bpt 0 = 8002022c
Bpt 1 = a002000c

PMON> b 80021248 -s "r"   Set a break with the command 'r'.
```

Environment

The command uses the brkcmd environment variable.

See Also

The db, load and boot commands.

boot

The **boot** command will load and start a program.

Command Format

```
boot [options] path [args]...
```

where:

path	path to the file to be loaded
args	arguments sent to the started program as argv

Functional description

The boot command is similar to the load command except that it also starts execution of the program that has been loaded.

The command can read raw binary files, S-Record files and files in the ELF format as used in:

- Algorithmics SDE-MIPS
- newer SGI compilers
- systems compliant with the MIPS/ABI standard
- older MIPS ECOFF format
- OpenBSD PowerPC, ARM and MIPS 32Bit ELF.
- Some Linux for PowerPC implementations

PMON2000 extracts any symbol table information from these files, and adds it to the target symbol table. Optionally the symbol table can be loaded after the loaded program image in a way suitable for for example OpenBSD and other operating systems built in debuggers. This function is activated by using the **-k** option.

The boot command normally clears the symbol table, exception handlers, and all breakpoints. The **-s**and **-b** options suppress the clearing of the symbol table and breakpoints, respectively. The value of the cpc register is set automatically to the entry point of the program. Therefore, to execute the downloaded program, only the g command is required. This is particularly useful when an mage has been loaded into flash ROM.

The boot command may return a large number of different error messages, relating to network problems or file access permissions on the remote host. For a file to be loaded via **TFTP** it **must be publicly readable**, and it may have to be in a directory which is acceptable to the remote server.

The **-f** (flash_load) option tells the boot wrapper to network load an image into the flash ROM area designated as an address argument. The area specified must be large enough to accept the image. The are specified will first be erased, then the image loaded and then verified. PMON2000 uses some temporary RAM to hold the image so it is important that the amount of free RAM exceed the size of the image (and the the flash area). This is rarely violated.

The boot command will detect cases where the program being loaded would overwrite PMON2000s crucial data or heap. Re-linking your program at a different (higher) address usually solves the problem.

Environment

The boot command uses bootfile environment variable.

See Also

The load command.

bt

The **bt** command unwind and display the stack

Command Format

```
bt [-v] [cnt]
```

where:

-v	verbose. Print any additional frame information
cnt	limit the nesting depth to ***cnt*** levels

Functional description

The **bt** command unwinds the debugged programs stack by inspecting the stack and code. The actual trace output is architecture dependent. The **-v** option is used to print out any available additional information like call arguments (register parameters) if this information is available. To limit the number of levels the traceback is going back the ***cnt*** argument can be used.

A typical stack trace output may look like this (MIPS architecture) after having reached a breakpoint:

```
PMON> c write+10
   write+0x0010     3c09a07f    lui        t1,0xa07f
PMON> bt
    write+0x0010    (0x00000001,0xa0030300,0x0000001c)
   flsbuf+0x0234    (0xa0030300,0xa0029030)
   printf+0x045c    (0xa0025490,0xa0020000,0x000000001,0x00000010)
     main+0x0138    (0x00000001,0xa07ffffe0)
   _start+0x0040    ()
```

Environment

The command uses no environment variable.

See Also

..

C

The **c** command resumes program execution after a breakpoint

Command Format

```
c [addr]
```

where:

addr	insert a temporary breakpoint at address

Invoking the **c** command without options resumes execution at the location given by CPC register.

Functional description

The **c** command resumes program execution at the location given by the **CPC** register. This command is normally used to resume execution after a breakpont is reached and execution has been stopped.

A temporary breakpoint may be specified by using the **addr** argument. The temporary breakpoint remains in effect only until the next time the program execution is halted by **any** breakpoint.

Examples illustrating the use of the **c** command follow:

```
PMON> c                 Resume executing at the current value
                        of the CPC register.

PMON> c a0020008        Set a temporary break at 0xa0020008 and
                        resume execution at the current value of
                        the CPC register.
```

Environment

The command uses no environment variable.

See Also

The go (g), trace (t) and trace over (to) commands.

call

The **call** command calls a function and print the return value.

Command Format

```
call addr [arg... | -s string]
```

where:

addr	address of function to call
arg	expression evaluated and passed as integer argument
-s *string*	passed argument is a pointer to ***string***. See below.

Functional description

The **call** command calls the function which address is given by the ***addr*** expression argument with the parameters given as **arg** and **-s str**. A maximum of five arguments can be used. String and integer values can be mixed in any order. String arguments containing spaces must be quoted.

When the called function returns the value returned is displayed in hex and decimal.

It can be useful to have dedicated printout functions in a debugged program that can be called with the call command to display complex data structures when debugging with PMON2000.

Environment

The command uses no environment variable.

See Also

The go (g) command.

cd

The **cd** command sets the current working directory.

Command Format

```
cd [path]
```

where:

path	new path to be used as current directory
..	go up one livel in the hierarchy

Functional description

The **cd** command sets the current working directory to *path*.

The argument .. moves the current directory path up one level unless the current path is already at the root level '/'.

Environment

The command uses no environment variable.

See Also

The pwd, dir and ls commands.

copy

The **copy** command copies one memory area from one addresss to another address.

Command Format

`copy` ***from to size***

where:

from	address where to copy from
to	address where to copy to
size	number of bytes to copy

Functional description

The **copy** command is used to copy an area of memory from one address to another address. If the source and destination areas overlap, copy will copy in the direction that will result in a non-destructive copy.

When the copy operation is finished any caches will be synchronized so any code copied will be cached in correctly by the processor caches.

Environment

The copy command uses no environment variable.

See Also

The fill command.

fill

The **fill** command fills a memory area with the given pattern.

Command Format

`fill` ***from to pattern***

where:

from	first address to fill
to	last address to fill
pattern	fill pattern to use when filling. See below

Functional description

The **fill** command is used to fill an area of memory with the given data pattern. The pattern is expressed as bytes and can be of any size not exeeding the maximum size of 80 bytes.

Examples of valid pattern arguments are:

```
0x13                        One character pattern.
0x13 0x10 0x0               A three character pattern.
0x13 -s "Hello world" 0x0   A pattern with a
                            zero terminated string.
```

The fill operation reuses the pattern until the last location has been filled.

Environment

The command uses no environment variable.

See Also

The search and copy commands.

flash

The **flash** command program available flash memory

Command Format

`flash [-qev] [-p` ***page***`]` ***flashaddr*** `[`***size***`[`***dataaddr***`]]`

where:

-q	query available flash devices
-e	erase flash
-v	verify flash
-p	when the flash is paged, selects the page
flashaddr	first address in flash to erase and/or program
size	size to erase and/or program
dataaddr	first address of data to program

Functional description

The **flash** command programs flash memory on the target board. The available flash memories can be queried by typing the flash command without any parameters or with the -q option. The output from the query is target system dependent but might look like:

```
PMON> flash -q
Available FLASH memory
 Start    Size      Width  Sectorsize  Type
ff000000 00800000  8*8    00040000    i28F160
fff00000 00080000  1*8    00010000    Am29F040
PMON>
```

The ***start*** address is where in the memory map the flash area starts. The ***size*** is the size in the memory map that this flash device occupies. This may be less than the actual flash size if the device is paged. The ***width*** indicates how many flash devices and what width each device has. The first value is the number of devices and the second is the width in bits of each device. The total word length can be determined by multiplying the width with the number of devices. The ***sector size*** is the minimum size that can be erased and reprogrammed without affecting any other data in the flash. Finally, the ***type*** indicates what type of devices the flash area is built of.

The **-e** option should be used to erase the entire flash or a part of a flash area. To erase the entire flash area only the start-address of the area has to be given. To erase a part of a flash area the start address and the desired size should be given. Remember though that the erase operation takes place in multiples of the sector size.

On some targets a flash memory can be **paged**. This means that the entire flash chip is not accessible at the same time but instead a part or **page** of the flash can be selected at a time. To select a page the command

```
PMON> flash -p <n>
```

where ***n*** is the desired page number should be used. If more than one flash area uses paged flash devices and each area has individual page selects the base address of the area must be given as well.

To program a flash area the flash command require three arguments, the ***flashaddr***, the ***size*** and the ***dataaddr*** arguments. The ***flashaddr*** argument determines where in the flash area the programming should start, the ***size*** argument determines the number of bytes that should be programmed and the ***dataaddr*** argument tells where the data to program into the flash is found. Unless the **-e** option is given programming is performed in 'patch' mode. Normally the **-e** option should be used to erase the flash before programming.

Some target systems have HW write protect features. When PMON2000 can determine whether a non-software controllable HW write protect is on or off it will display this information. If the HW write protect feature can be controlled by software PMON2000 will take the required action to turn off write protection before programming and turn on protection again after programming.

Environment

The command uses no environment variable.

See Also

...

flush

The **flush** command flush the processor caches

Command Format

```
flush [-id]
```

where:

-i	flush the instruction cache
-d	flush the data cache

If the flush command is used without any arguments both the instruction cache and the data cache will be flushed.

Functional description

The **flush** command invalidates the instruction cache and typically writeback the data cache.

Environment

The command uses no environment variable.

See Also

…

g

The **g** command start program execution.

Command Format

```
g[-st] [-b addr] [-e addr] [-- args...]
```

where:

-s	don't set client stack pointer
-b *addr*	insert a temporary breakpoint at address ***addr***. This breakpoint is removed next time execution halts.
-e *addr*	start execution at address ***addr***. This breakpoint is removed next time execution halts.
- - *args...*	indicates that the argument or arguments ***args*** are to be passed to the client program. No more options to the g command itself can be given after this option

Invoking the **g** command with no options starts execution at the location given by **CPC** register.

Functional description

The **g** command starts program execution. If the user does not specify a starting address, execution starts at the current value of the **CPC** register. This command must only be used once after downloading a new program. Use the c command to continue execution after a breakpoint.

If the user specifies the **--** option, then PMON2000 will pass all arguments after -- to the client program. See Program run environment for details on argument passing.

A temporary breakpoint may be specified by using the **-b addr** argument. The temporary breakpoint remains in effect only until the next time that program execution is halted.

When starting a program the called function main will receive four incoming arguments, argc, argv, envp and pmonvecp. The first argument will be the string 'g' to distinguish this start method from the other. Then any args given after the **--** switch will follow.

The **-e *addr*** option causes PMON2000 to start execution at the address ***addr*** instead of the **CPC**value.

Examples illustrating the use of the g command:

```
PMON> g                         Start executing at the current
                                value of the CPC register.

PMON> g -e a0020000              Start executing at 0xa0020000.

PMON> g -e a0020000 -b a0020008  Start executing at 0xa0020000 and
                                 break at 0xa0020008.
```

Environment

The command uses no environment variable.

See Also

The continue (c) command.

h

The **h** command provides a built-in help function.

Command Format

```
h [*|cmd...]
```

where:

*	displays detailed help about all commands
cmd	displays detailed help about listed commands

If the command is executed without any parameters, then a short list of all the available commands is displayed.

Functional description

The **h** command provides built in help. If used without arguments, all available commands are listed. Used with one or more command names as an option, it displays a detailed help on those commands.

The "*****" option produces detailed help on all the commands, using the more command to paginate output to the screen.

Examples illustrating the use of the h command follow. NOTE that the actual output is architecture and configuration dependent.

```
PMON> h
                                  Shell
       h  on-line help                    about  about PMON2000
      sh  command shell                      hi  display command history
    more  paginator                        stty  set tty options
    vers  print version info               eval  evaluate and print result
    date  get/set date and time
                               Environment
     set  display/set variable             eset  edit variable(s)
   unset  unset variable(s)
                              Boot and Load
    load  load memory from hostport        boot  boot wrapper
 netboot  load file via network        diskboot  boot from disk
                                Debugger
       g  start execution (go)                c  continue execution
       t  trace (single step)                to  trace (step over)
      db  delete break point(s)               b  set break point(s)
     sym  define symbol                      ls  list symbols
       r  display/set register                l  list (disassemble) memory
      bt  stack backtrace
                                 Memory
       m  modify memory                       d  display memory
    copy  copy memory to memory            fill  fill memory
  search  search memory                      mt  memory test
                                  Misc
      tr  transparent mode                 dump  dump memory to hostport
   flush  flush caches                     call  call function
  reboot  reboot system                    ping  ping remote host
   flash  program flash memory
                                   Pci
    pcfg  pci config space
PMON>
PMON> h stty
 stty [tty] [-va] [baud] [sane] [term] set terminal options
```

Environment

The command uses no environment variables.

See Also

The about command.

hi

The **hi** command list the history buffer.

Command Format

```
hi [cnt]
```

where:

cnt	list the last ***cnt*** commands in the history buffer

Entering the command with no parameters list the last up to 200 command lines.

Functional description

The **hi** command shows the command history, together with the history number for each command, in reverse order (the last command entered is listed first; the first command entered is listed last). The history and command numbers are reset to zero each time the system is reset.

Entering the **hi** command with no arguments lists the last 200 commands. This option is useful for determining the history number for a particular command.

The user can page through the output of the **hi** command, one screen at a time.

The optional ***cnt*** parameter selects a set number of lines to be output. The history list is intentionally in the reverse order to that used in a C shell, so that the latest entry is displayed first. If a command line is identical to the previous command, it is not added to the command history.

Examples illustrating the use of the hi command follow.

```
PMON> hi 3         Display the three last commands.

14 hi 3
13 hi
12 l


PMON> hi  Display the entire history, using more

13 hi     to control the screen output.
12 l
11 to
10 t
9  l
8  g start main
7  hi
6  g
5  ls -a @pc
4  d start+200+0t13*4

more-(q)
```

Environment

The command uses the more function and the moresz environment variable.

See Also

The sh command which maintain a command history.

ifaddr

The **ifaddr** command configures a network interface.

Command Format

ifaddr ***ifname ipaddr***[***:ifparameters***] ifaddr ***ifname*** bootp

where:

ifname	name of the interface
ipaddr	ip address assigned to the interface
ifparameters	optional configuration parameters

Functional description

The **ifaddr** command is used to configure IP-addresses on interfaces accessible within PMON. The interface can be configured using parameters such as ipaddress, netmask, gateway etc directly or via the bootp protocol.

To configure an interface using the bootp protocol the command:

```
PMON> ifaddr fxp0:bootp
```

is used. PMON2000 will then issue a bootp request to a bootpserver available on the same network as the interface and configure the interface with the given parameters. A set of environment variables will also be set which describes the configuration.

When configuring the interface manually the minimum required parameters is the interface name and an IP address such as:

```
PMON> ifaddr fxp0:192.168.16.10
```

By adding ':' separated additional parameters netmask, broadcast and default gateway can be specified. The format of the config string is:

```
interface:ipaddress:netmask:broadcast:gateway
```

An example:

```
PMON> ifaddr fxp0:192.168.16.10:255.255.255.0::192.168.16.1
```

Note that when a parameter is omitted, like the broadcast address in the example, the field should be left empty by just typing the ':' separator. Unless a specific need for a non-standard netmask or broadcast address is requiered these fields are typically left empty. Netmask and broadcast address is calculated from the IP address.

Environment

The command uses no environment variable.

See Also

The ifconfig environment variable.

l

The **l** command disassembles and display instructions in memory.

Command Format

```
l [-bc] [addr[cnt]]
```

where:

-b	list only branch instructions
-c	list only call instructions
addr	address where disassembly should start
cnt	disassemble cnt instructions and then stop

Invoking the l command with no options starts disassembly from the location given by CPC until quit from 'more' is done.

Functional description

The **l** command is used to examine memory by disassembling machine code into human readable text. If a starting address is not specified, disassembly starts at the current value of the **CPC** register. Otherwise the disassembly starts at the address given in the ***addr*** argument. The output is sent through 'more' and quitting more terminates the command. The ***cnt*** argument can be used to disassemble ***cnt*** number of instructions and then stop. When the ***cnt*** argument is given the output is not sent through 'more' but all lines are output until all lines have been printed.

On some targets the **l** command can be told to only list branch and/or call instructions. This is useful when the program flow is to be examined. The **-b** option is used to select branches while the **-c** option select calls.

The output of the disassembly command is highly target architecture dependent.

Environment

The command uses no environment variable.

See Also

The dump (d) and more commands.

load

The **load** command loads a program into memory but does not start it.

Command Format

`load [-options]` ***pathname***

where:

-f *addr*	load into flash at address ***addr***
-o *offset*	offset ***offset*** is added to the load adress
pathname	url or direct pathname to the file to be loaded

Functional description

The load command is similar to the boot command except that it does **not** start execution of the program that has been loaded. Instead it returns to the PMON2000 prompt awaiting the next command.

The command can read raw binary files, S-Record files and files in the ELF format as used in:

- Algorithmics SDE-MIPS
- newer SGI compilers
- systems compliant with the MIPS/ABI standard
- OpenBSD ARM, PowerPC and MIPS 32Bit ELF.
- Some Linux for PowerPC implementations

PMON2000 extracts any symbol table information from these files, and adds it to the target symbol table. Optionally the symbol table can be loaded after the loaded program image in a way suitable for for example OpenBSD and other operating systems built in debuggers. This function is activated by using the **-k** option.

The boot command normally clears the symbol table, exception handlers, and all breakpoints. The **-s**and **-b** options suppress the clearing of the symbol table and breakpoints, respectively. The value of the CPC register is set automatically to the entry point of the program. Therefore, to execute the downloaded program, only the **g** command is required.

The boot command may return a large number of different error messages, relating to network problems or file access permissions on the remote host. For a file to be loaded via **TFTP** it **must be publicly readable**, and it may have to be in a directory which is acceptable to the remote server.

The **-f** (flash_load) option tells the boot wrapper to network load an image into the flash ROM area designated as an address argument. The specified area must be large enough to accept the image. The specified area will first be erased, then the image loaded and then verified. PMON2000 uses some temporary RAM to hold the image so it is important that the amount of free RAM exceed the size of the image (and the the flash area).

The **-o*offset*** option provides for assigning a transfer and run (execute) address header for the image that is being loaded into flash ROM. This option is critical when the image being loaded into flash ROM is, in fact, destined to be executed later out of RAM at another address. PMON2000 itself uses this feature because it is stored in flash ROM (usually) at one address, but moved and run from a RAM address on most platforms. The **-o *offset*** value specifies the actual address where execution is to begin AFTER an image is transferred by the **g** command.

Boot an image into Flash ROM Example:

```
PMON> load -f ff000000 -o 10000 tftp://loke.opsycon.se/pmon.db64360
Loading file: tftp://loke.opsycon.se/pmon.db64360 (elf)
0x100074/447112 + 0x16d2fc/42508(z)
Programming flash 0x00100000:0x00077894 into 0xff000000
Erasing FLASH block  0 Done.
Erasing FLASH block  1 Done.
Programming FLASH. Done.
PMON>
```

In the above example, the **boot -f** command was used to load a new PMON2000 image into the 512Kb onboard flashROM on the Marvel DiscoveryII evaluation board. The image was resident on a local network server, in the /tftpboot directory under the name of pmon.db64360.elf and was a standard PowerPC ELF-32 image created with GNU tools on an OpenBSD PowerPC system. The image file is loaded and we get to see the binary file characteristics echoed to the console while the load is happening. The **load -f** command also informs us of the transfer address (0x00100000:0x00077894) and size, as well as confirmation of the stored location (into 0xff000000). Next it tells us which flash ROM block(s) are being erased to hold the new image, followed by confirmation that the flash ROM is being programed, and then reports its success.

When reading the symbol table PMON2000 may complain that it does not have enough room to store the programs symbols. To increase the size of the heap, use the set heaptop command to reserve more space and, if necessary, re-link your program with a higher base address. The boot command will also detect cases where the program being loaded would overwrite PMON2000s crucial data or heap: again re-linking your program at a different address will cure the problem.

Environment

The command uses the bootfile environment variable.

See Also

The boot command and url/pathname description.

ls

See the dir command.

The **lsym** command list symbols from the loaded symbol table.

Command Format

```
ls [-ln] [-a addr] [symbol]
```

where:

-l	list in long form with values
-n	list in numerical (value) order. Default is in name order
-a *addr*	find symbol closest to ***addr***
symbol	lookup and display value for ***symbol***. ***symbol*** can specify 'wildcard', see below.

Functional description

The **lsym** command display information from the loaded symbol table. The **-l** switch lists the symbols values along with their names. The **-n** switch changes the sort order from symbol names to symbol values.

Wildcard values can be used with the ***symbol*** name. Valid wildcard characters are '?' and '*'. The '?' matches any single character at that position while the '*' character matches any number of any characters. See the examples for more information.

When using the **-a *addr*** switch the symbol table will be searched for the symbol with the value closest to the addr expression. If no symbol is found within **20KB** of the given value no symbol is found.

Examples:

Define a couple of symbols using the sym command.

```
PMON> sym sym 0x1000
PMON> sym syx 0x2000
PMON> sym symbol 0x23800
```

List the entire symbol table in short form.

```
PMON> lsym
 Pmon _ftext etext start sym symbol syx

PMON> lsym -n
 sym syx symbol etext Pmon _ftext start
```

List the entire symbol table in long form.

```
PMON> lsym -l
 80010000 Pmon
 8008cd10 _ftext
 07f80000 etext
 80100000 start
 00001000 sym
 00023800 symbol
 00002000 syx

PMON> lsym -ln
 00001000 sym
 00002000 syx
 00023800 symbol
 07f80000 etext
 80010000 Pmon
 8008cd10 _ftext
 80100000 start
```

List the symbols matching the wildcard specification. Note how the wildcard specification narrow down the selection and targets the desired result.

```
PMON> lsym s*
 start sym symbol syx

PMON> lsym -l s*
 80100000 start
 00001000 sym
 00023800 symbol
 00002000 syx

PMON> lsym -l sy*
 00001000 sym
 00023800 symbol
 00002000 syx

PMON> lsym -l sy?
 00001000 sym
 00002000 syx

PMON> lsym -l sym*
 00001000 sym
 00023800 symbol

PMON> ls -l sym?*
 00023800 symbol
```

Lookup an address in the symbol table.

```
PMON> lsym -a 0x23834
 00023834 = symbol+0x34
```

Environment

The command uses no environment variable.

See Also

The sym commands.

m

The **m** command modifies memory contents.

Command Format

```
m [-bhwd] addr [data... | -s string]
```

where:

-b	store 8 bit values
-h	store 16 bit values
-w	store 32 bit values
-d	store 64 bit values (on supporting architectures)
addr	address where to start storing data
data	data to be stored
-s *string*	string argument, stored as bytes

Invoking the **m** command with only the addr argument enters the interactive command mode.

Functional description

The **m** command is used to modify memory. Modification can be done in two ways, directly or interactive. In direct mode the data is given on the command line and the command terminates when all the data items given has been stored in memory. When no data is given on the command line the interactive mode is entered and the user can interactively display and modify memory locations.

If no size option is given to the command the environment variable datasize will be used to determine the size of data to work with.

When using a string as the data argument, **-b** will be the default datasize regardless of what the datasize environment variable is set to. Data is then taken from the given string argument and stored in consecutive locations. When **-s** option is used only the direct mode of the command is usable.

The following commands can be used when using the interactive mode.

Key	Action

=	reread the current location
<cr>	next address
-	previous address
^	previous address
.	exit interactive mode

Environment

The command uses the datasize environment variable.

See Also

The dump (d) command.

more

The **more** command paginates console output.

Command Format

The **more** command is a builtin function and can not be invoked from the command line.

Functional description

The more command is not specified by the user on the command line, but is implicitly used by most commands. After displaying the number of lines according to the value of the moresz environment variable, the more command displays the prompt "more-".

The user can enter the following commands at the "more-" prompt:

Key	Action
Space	display next page
/*str*	seach forward for *str*
n	repeat last search
<cr>	display one more line
q	quit more and terminate command

Environment

The command uses the moresz environment variable to determine how many lines to display at at time. If moresz is set to zero, the screen scrolls continuously. The ^S or ^Q control sequence can be used to pause the output, and the ^C control sequence may be used to terminate output before the command finishes by itself.

See Also

The set command and environment for environment variable settings.

mt

The **mt** command performes a simple memory test.

Command Format

```
mt [-cv] [start [size]]
```

where:

-c	continue (loop) until stopped
-v	verbose. Show progress
start	start address of area to test
size	size of area to test in bytes

Functional description

The **mt** command is used to do a simple memory test. The test of the memory is simply done by writing each bit in every word and by writing each word locations address with its address and then check that the address can be read back from all locations. To have a slight chance to detect address errors all addresses are first written and then read back. It is not intended to do any rigorious memory testing but instead be a test that can be used to check that installed memory modules comes up as PMON2000 probed them and detect possible problems that has to be further investigated.

Environment

The command uses no environment variable.

See Also

...

pcicfg

The **pcicfg** command access PCI configuration space.

Command Format

`pcicfg` ***bus device subfunc register***

where:

bus	PCI bus to access
device	PCI device to access
subfunc	PCI subfunction to access
register	PCI register to access

Functional description

The **pcicfg** command is used to access the PCI configuration space in the given bus. The command takes four arguments, the bus number, the device number, the subfunction number and the first register to access. The register number must be 32-bit aligned and all accesses are made as 32-bit read and writes. When executing the command an interactive mode is entered. The register number along with the current contents of the configuration register is displayed. The user can then enter one of the following commands:

Item	Action
expression	value to be stored
=	reread the current location
<cr>	forward to next address
-	back to previous address
^	back to previous address
.	exit interactive mode

If an expression is given, the register displayed will be written with the new value and the next register will be displayed.

Values are always displayed and entered in little endian or the native PCI endian. What this mean is that bytes do not have to be swapped if the target is a big-endian target.

Environment

The command uses no environment variable.

See Also

The pciscan command.

pciscan

The **pciscan** command scan for devices on PCI bus.

Command Format

```
pciscan [bus]
```

where:

bus	PCI bus to scan

Functional description

The **pciscan** command scan and displays information about devices found on the PCI bus.

Environment

The command uses no environment variable.

See Also

The pcicfg command.

ping

The **ping** command test network connectivity.
Command Format
`ping [-fqv][-c n][-s sz][-i sec][-l n] host`
where:

-f
flood ping. Send packages as fast as possible

-q
?

-c *n*
send ***n*** packages and stop

-s *sz*
size of packet expressed in bytes

-i *sec*
wait ***sec*** seconds before sending next packet

-l *cnt*
presend ***cnt*** packages before falling back to interval sending

host
target to where packets are sent

Functional description

The ping command sends ICMP_ECHO_REQUEST packages to the given remote host and monitors any ICMP_ECHO_RESPONSE packages received back. This command is useful to test conectivety with a remote host.

The **-f** flag can be used to flood ping the remote host. Packages are then sent as fast as possible. Note that using this option may lead to network congestion and should be used with care on networks shared with others.

When using ping for fault isolation, start by pinging 127.0.0.1 (a universal self-address, by internet convention.) This verifies that at least the onboard setup is workable. Then, hosts and gateways further and further away should be pinged. Round-trip times and packet loss statistics are computed. If duplicate packets are received, they are not included in the packet loss calculation, although the round-trip time of these packets is used in calculating the minimum/average/maximum round-trip time numbers. When the program is terminated by a ^C a brief summary is displayed.

Ping will report duplicate and damaged packets. Duplicate packets should never happen: they have to be gateway problems. Tell your network manager.

Damaged packets (data doesnÂ't look like it should) are serious cause for alarm and often indicate broken hardware; somewhere in the ping packets path (in the network or in the hosts).

Environment

The command uses no environment variable.

See Also

...

pwd

The **pwd** displays the current working directory path.

Command Format

```
pwd
```

Functional description

The **pwd** command displays the current working directory path.

Environment

The command uses no environment variable.

See Also

The pwd, dir and ls commands.

r

The **r** command set and display register values.

Command Format

```
r [reg [value | field value]]
```

where:

reg	specifies a register to display/set
value	value expression to set register to
field value	a field value, usually with special registers

Functional description

The **r** command display and sets register value. Typing only the **r** command displays the most common registers while the command **r *** display all registers. To display a particular register only the name of that register can be given as argument. For example, the command **r r10** display the contents of register r10.

Floating point registers (if available) can be displayed by using the **f*** or **f<n>** register specification.

When assigning new values to registers a specific register must be specified along with an expression giving the new register values.

Some targets allow assignment to specific fields in typically, control, configuration and status registers.

Environment

The command uses no environment variable.

See Also

…

reboot

The **reboot** command restarts the target.

Command Format

```
reboot
```

The **reboot** command has no parameters.

Functional description

The **reboot** command is used to restart PMON2000 as if the reset button was activated. The exact type of reset/restart is determined by the way software reset is implemented in the target hardware.

When the reboot command is executed all settings and temporary variables will be lost and all actions like autoboot etc will be executed as if a panel or power on reset was issued.

It should be noted that on some targets the reboot command is not fully reliable since the software controlled reset is not properly resetting the target. PMON2000 attempts to do the best of the situation though.

Environment

The command uses no environment variable.

See Also

…

search

The **search** command searches the memory for patterns.

Command Format

`search` ***from to pattern***

where:

from	address where search should start
to	address where search should end
pattern	pattern specification. See below on how to specify patterns

Functional description

The **search** command is used to search an area of memory for the given data pattern. The pattern is expressed as bytes and can be of any size not exeeding the maximum size of 80 bytes.

Examples of valid pattern arguments are:

```
0x13                         One character pattern.
0x13 0x10 0x0                A three character pattern.
0x13 -s "Hello world" 0x0   A zero terminated string.
```

The search operation is terminated when the ***to*** address is reached. Each time the pattern is found the memory locations containing the pattern is displayed.

The following example searches the first megabyte of memory for the NULL terminated string PMON and display all occurances.

```
PMON> search 0 100000 -s PMON 0x0
0000fcfc  504d4f4e 00000000  00000001 000871c8   PMON.........q
0006739a  504d4f4e 00000000  00000a50 4d4f4e2f   PMON.......PMON
0007f140  504d4f4e 00307830  00000000 00000000   PMON.0x0.......
00087ee3  504d4f4e 0000087e  e8000000 03682073   PMON...~.....h
PMON>
```

Environment

The command uses no environment variable.

See Also

The copy and fill commands.

set

The **set** command set and display environment variables.

Command Format

```
set [-t] [name [value]]]
```

where:

-t	set temporarily. Setting is not stored in non-volatile memory
name	environment variable name
value	string value to set variable to

Functional description

The **set** command is used to set or display environment variable values. Environment variables are normally stored in NVRAM. The **-t** switch can be used to not store the value in NVRAM. The setting will be lost when PMON2000 is restarted. This may be useful when a variable need to be set but does not need to be saved.

Some variables can only be set to predefined values. When such a variable is displayed PMON2000 also displays the variables list of allowed values enclosed in square brackets. No list is shown if the variable can have any value. In general, when the value is a numeric value, or when the value has an unlimited range of possible values, no list is shown.

The set command does not evaluate the specified value but check constrained variables against a list of allowed values. Value checking on other variables are only performed when a command uses a variable.

To set a variable to a multiple-word value, enclose the value in single or double quotation marks.

The maximum length for a variable name and its value must be less than 255.

Examples illustrating the use of the set command follow.

```
PMON> set                            Display all current values.
    brkcmd = "l @cpc 1"
    datasz = -b           [-b -h -w]
   inalpha = hex          [hex symbol]
    inbase = 16           [auto 8 10 16]
    moresz = 10
  regstyle = sw           [hw sw]
    rptcmd = trace        [off on trace]
   trabort = ^K
     uleof = %
      ulcr = off          [off on]
   validpc = "_ftext etext"
   heaptop = 80020000
    dlecho = off          [off on lfeed]
   dlproto = EtxAck       [none XonXoff EtxAck]
  hostport = tty1
    prompt = "PMON> "
 etheraddr = 12:34:56:78:90:ab
    ipaddr = 71.0.0.211
   vxWorks =
      diag = 0            [N[:dev]]
```

```
PMON> set moresz                     Display current moresz.
moresz = 10

PMON> set moresz 20                  Set moresz to 20 decimal.
```

Environment

The command changes environment variables.

See Also

The unset and eset commands.

sh

The **sh** command is the builtin command shell.

Command Format

The **sh** command is a built-in function and can not be invoked from the command line.

Functional description

The sh command is not user invoked command but is implicitly run as the command shell when PMON2000 enters interactive mode.

Command prompt

When the command shell is ready to execute a new command, it outputs the default command prompt:

```
PMON>
```

The user can change the string output as the prompt by setting the environment variable prompt to the desired string value.

```
PMON> set prompt "My very own pmon prompt> "
My very own pmon prompt>
```

The metacharacter **'!',** when used in the prompt string, will be replaced with the command history number assigned to the next command

Command line editor

PMON2000 have a built-in command line editor which can be used to edit the command line while typing. It can also be used to edit a previously entered command and reissue it in its edited version. See the hi command for information about command history.

The edit commands are:

up-arrow or ^P	Recall the previous command. Typing repeated up-arrow or ^P's recalls the next command from the command history.
down-arrow or ^N	Recall the next command. Typing repeated down arrow or ^N's recalls the next command from the command history.
!str	Recall the first occurence of command starting with str from the command history and execute it.
!num	Recall the command numbered num from the command history and execute it.
!!	Recall the last command from the command history and execute it.

<cr>	Repeat the last command if repeateable. See rptcmd.
right-arrow or ^F	Move the cursor forward one position
left-arrow or ^B	Move the cursor bacwards one position
'home' or ^A	Move the cursor to the beginning
'end' or ^E	Move the cursor to the end
^D	Delete the character under the cursor
^H	Delete the character before the cursor

Variable substitution

When entering commands, environment variables can be called by typing their name preceded by a **$**sign. Variable names may be enclosed in **{}** to make them stand out. This is useful to get expansion of ${myvar}foo to work properly while $myvarfoo will not expand. Also, if the variable name contains other characters than **A-Z, a-z** and **0-9** it must be enclosed in **{}**.

To input a $ sign without expansion $$ should be typed.

The maximum length a command line is allowed to expanded to is limited to 512 characters.

Variable substitution can not be nested in the current version of PMON2000.

Multiple commands

More than one command can be entered on one command line by separating them with a semicolon character. This is useful when setting a variable to execute a sequence of commands. Example:

```
PMON> set mycmd "boot boothost:myprog; b testfunc; g start"
PMON> $mycmd
```

Note that the quote characters **'** and **"** will cause the the embedded string to be treated as a single entity. However, one level of quotes will be stripped of each time the string is evaluated.

Aborting a command

Typing **^C** aborts the current command and returns to the PMON2000 shell prompt. Note that this is also the case if PMON2000s I/O routines are used for input and output in separately executed program.

Environment

The command uses several environment variables which control the behaviour of the command shell. See the environment manual section.

See Also

The more, hi and set commands.

The environment and expression reference on environment usage and expression construction.

stty

The **stty** command set and display terminal settings.

Command Format

```
stty [device][-a] [baud][sane][flags]
```

where:

device	name of the device to change
-a	show all tty parameters
baud	set port to the baud baudrate
sane	reset settings to default, leaving badrate as is
ixany	allows any character to restart the output
-ixany	allows only START to restart the output
ixoff	enables tandem mode
-ixoff	disables tandem mode

When invoking the **stty** command with no parameters, the terminal type and baud rate for the tty0 port is displayed

Functional description

The stty command displays and sets the terminal options, such as terminal emulation type, baud rate, and ioctl settings.

```
PMON> stty
baud=9600
PMON> stty -a
istrip ixon -ixany -ixoff icanon echo echoe icrnl onlcr
erase=^H stop= start=^Q eol=^J eol2=^C vintr=^C
```

Although the stty command allows all listed flag settings to be changed the expected effect on the line dicipline on the console port may not be achived. This is due to the built-in line edit capability. Setting or clearing these parameters on other tty ports will have the desired effect though.

Environment

The command uses no environment variable.

See Also

…

sym

The **sym** command define and enter a symbol to the symbol table.

Command Format

`sym` ***name value***

where:

name	symbol name
value	expression value to be assigned

Functional description

The **sym** command can be used to enter a symbol into the symbol table for later referral when debugging. The symbol will be given the name **name** and the value from the expression **value.**

Examples:

```
PMON> sym newstart start+0x100
PMON> sym myconstant 0x23800
```

Environment

The command uses no environment variable.

See Also

The lsym command.

t

The **t** command step program execution when debugging.

Command Format

```
t [-vbci][-m addr val][-M addr val][-r reg val] [-R reg val][cnt]
```

where:

-v	verbose. Display each step when cnt is given
-b	verbose. Display only branches when cnt is given
-c	verbose. Display only function calls when cnt is given
-i	stop on invalid CPC value
-m *addr val*	stop when memory location ***addr*** becomes equal to ***val***
-M *addr val*	stop when memory location ***addr*** becomes not equal to ***val***
-r *reg val*	stop when register ***reg*** becomes equal to ***val***
-R *reg val*	stop when register ***reg*** becomes not equal to ***val***
-i *cnt*	trace ***cnt*** instructions and then stop

Functional description

The **t** command resumes program execution at the current value of the **CPC** register and halts when one instruction has been executed.

The **t** command will be repeated ***cnt*** times if a value is given. If the **-v**switch is given the command determined by the enviroment variable brkcmd will be executed each time. Otherwise the brkcmd will only be executed the last time when program execution is halted.

The **-m, -M, -r** and **-R** options can be used to stop execution when a memory location or a register becomes equal or not equal to the given value. If a breakpoint is reached or any other stop condition occurs before the equal or not equal condition program execution will halt.
The ***cnt*** argument can not be used when match/no match is used.

Environment

The command uses the brkcmd environment variable.

See Also

The go (g), continue (c) and to commands.

to

The **to** command step program execution when debugging.

Command Format

```
to [-vbci][-m addr val][-M addr val][-r reg val] [-R reg val][cnt]
```

where:

-v	verbose. Display each step when cnt is given
-b	verbose. Display only branches when cnt is given
-c	verbose. Display only function calls when cnt is given
-i	stop on invalid CPC value
-m *addr val*	stop when memory location ***addr*** becomes equal to ***val***
-M *addr val*	stop when memory location ***addr*** becomes not equal to ***val***
-r *reg val*	stop when register ***reg*** becomes equal to ***val***
-R *reg val*	stop when register ***reg*** becomes not equal to ***val***
-i *cnt*	trace ***cnt*** instructions and then stop

Functional description

The **to** command resumes program execution at the current value of the **CPC** register and halts when the **location after the current location** is reached. Eg if the instruction to be executed is a branch or call instruction the next stop is when the branch is not taken or the called function returns. This is very useful when stepping through program loops and subroutine calls and it is not desired to single step the loop or subroutine.

The **to** command will be repeated ***cnt*** times if a value is given. If the **-v**switch is given the command determined by the enviroment variable brkcmd will be executed each time. Otherwise the brkcmd will only be executed the last time when program execution is halted.

The **-m, -M, -r** and **-R** options can be used to stop execution when a memory location or a register becomes equal or not equal to the given value. If a breakpoint is reached or any other stop condition occurs before the equal or not equal condition program execution will halt.
The ***cnt*** argument can not be used when match/no match is used.

Environment

The command uses the brkcmd environment variable.

See Also

The go (g), continue (c) and t commands.

tr

The **tr** command connect the console port with the host port.

Command Format

```
tr
```

The **tr** command has no parameters.

Functional description

The **tr** command is used to connect the PMON2000 console to the host port. The host port is given by the environment variable hostport. The host port can not be the same port as the PMON2000 console port.

When the tr command is started PMON2000 displays the line:

```
Entering transparent mode, ^K to abort
```

The characters typed on the keboard will be sent to the host port and characters received from the hostport will be displayed on the console. The abort character, ^K, can be changed by setting the environment variable trabort to the character that should end transparent mode.

Environment

The command uses the hostport and trabort environment variables.

See Also

The stty command.

unset

The **unset** command removes or resets an environment variable.

Command Format

```
unset name
```

where:

name	name of variable to remove or reset

Functional description

The **unset** command unsets an environment variable. If the variable is one of the system variables and has a default value the variable will be set to the default value.

Environment

The command changes the environment variables.

See Also

The set, eset and resetenv commands.

vers

The **vers** command display version information.

Command Format

```
vers [-a]
```

where:

-a	show all version information

Functional description

The **vers** command can be used to display information about the PMON2000 version and depending on the target platform, information about various platform specific versions.

When using the **-a** switch, target version information will be displayed. The information varies from target to target and can for example be board revision level, PLD revision level etc.

Examples:

```
PMON> vers
PMON: 1.0 Build 1.37
PMON> vers -a
PMON: 1.0 Build 1.37
Board revision level: B
PLD revision levels: 1.2 and 1.0
PMON>
```

Environment

The command uses no environment variable.

See Also

The about command.

The tftp server

PMON2000 R3.x can be configured to provide tftp server functionality to upload and download files to the onboard flash or filesystem.

Starting the server

The built in tftp server can be started automatically or from the command prompt. When started automatically it will be run in the background servicing requests from a client to upload or download files. For example, the autoboot sequence can be set to go into tftp mode if no loadable file can be found on the target system thus make an initial setup to work without having access to the console on the target. When in this mode the target can set up to announce its existens by broadcasting a special packet.

Upload and execute code

This method can be used to upload code to the ram memory and execute it. This works like a push load. The client put up a file to the target to the **/dev/ram/usermem** destination. The file is stored in RAM and when the upload has completed the file integrity is verified by checking the CRC32 which should be appended to the file. A tool for Linux and BSD called **prepimg** is available in the PMON2000 source tree in the tools directory. If the CRC32 is correct the uploaded file is checked for ELF or binary format. An ELF format file is loaded the same way as from a local media on the target itself. A binary file is executed in the same place as it was loaded and must either be linked to start at the proper location or be in PIC. Additionally, a signature can be used as a key to verify the uploaded file. The signature is 64-bit and is stored in the **signature** environment variable.

Upload and program flash

To upload a file a program it to flash the destination should be **/dev/flash@addr** where addr is an offset relative to the configuration definition of **USERFLASH_BASE**. The file is stored in RAM and when the upload has completed the file integrity is verified by checking the CRC32 which should be appended to the file. A tool for Linux and BSD called **prepimg** is available in the PMON2000 source tree in the tools directory. If the CRC32 is correct the uploaded file is programmed into flash and then the CRC32 of the programmed data is verified. The **signature** can also be used here to verify the file type.

Upload and save

All other destination for an upload will be to a filesystem on the target.

Environment

This function uses the signature environment variables affect this function.

See Also

Variables

Environment variables

Environment variables are used to control the behavior of PMON2000 and pass information between programs. Some variables affect console input and output, other controls the default settings for commands while some controls the boot process such as boot device and automatic startup.

To set, change or clear variables the set, unset, eset and resetenv commands are used.

Environment variables are usually stored in the targets NVRAM. The maximum length of an environment variable is 254 characters, name and value together.

PMON2000 variables

The following variables are used by PMON2000 to control its basic behavior:

Variable	Default	Constraints	Description
prompt	PMON>	<max string>	This variable can be used to change the standard PMON Shell prompt. The meta character '**!**', when used, will be replaced with the command sequence number.
moresize	10	<integer>	This variable specifies how many lines to display during screen-at-a-time display. See the more command.
inalpha	hex	[hex symbol]	This variable selects whether strings starting with the ASCII characters a, b, c, d, e, and f are interpreted as symbols or hexadecimal numbers. See expressions.
inbase	16	[auto 8 10 16]	This variable selects the default input base for numeric values. Users can input octal, decimal, or hexadecimal numbers by changing this variable. See expressions.
datasize	-b	[-b -h -w -d]	This variable controls whether data is displayed in byte, half-word, word or quad groups. See the d command.
rptcmd	trace	[off on trace]	When set to "on," the previous command is repeated when the user enters an empty line.

			When set to "trace" only trace commands will be repeated. See the sh command.
trabort	^K	<char>	This variable selects the character that terminates transparent mode and returns to command mode. See the tr command.
TZ	<none>	<string>	When set to GMT+-<n> the time displayed by the date command will be adjusted accordingly.

Networking

The following variables affect networking behavior:

Variable	Default	Constraints	Description
ethaddr	as set by mfg.	ethernet addr	This variable is used to specify the hardware Ethernet address on platforms which does not provide a way to store this value with the actual HW function. It should not be used for any other purpose. It is not used or available on platforms not having need for it. See the section on Ethernet specifics for further information.
ifconfig	<none>	<interface configuration string>	This variable is used to automatically configure any network interfaces during bootup. The variable is set to ';' separated strings, one for each interface, that should be configured. The first interface set up will get the default route. For information about how to set up the string see the ifaddr command.
localdomain	<none>		This variable should be set to the name of the local domain when using a nameserver.
nameserver	<none>		This variable is set to a whitespace separated list of nameserver ip addresses to enable the nameserver lookup function.

Booting & dumping

The following variables affect booting:

Variable	Default	Constraints	Description
autoboot	<none>	<max string>	This is the command which will be executed to autoboot a system. If this variable is set PMON will execute it after the number of seconds specified in bootdelay have elapsed. If the variable is not set, PMON2000 will enter the command shell.
bootdelay	<none>	<integer>	Specifies the delay in seconds before autoboot is executed.
bootfile	<none>	<pathname>	If no filename is given to the boot command the value set to this variable, if set, will be used as the filename.
dlproto	EtxAck	[none EtxAck XonXoff]	Specifies the download protocol used when loading via serial port.
hostport	tty0	<loadable port>	This variable will be used to select the port to be used when downloading is done using the oload command.
dlecho	off	[off on lfeed]	This variable controls whether the target board echoes on downloads. An entire line can be echoed, a single linefeed character can be echoed, or there can be no echo at all. See the load command and the section on flow control.
ulcr	cr	[cr lf crlf]	This variable defines whether there is a carriage return or both a carriage return and a linefeed character at the end of the line during dumps. See the dump command.
uleof	%	<max string>	This variable specifies a string that is sent to the host after a dump to the target has completed. See the dump command.

Debugging

The following variables affect debugging:

Variable	Default	Constraints	Description
brkcmd	l -r @cpc 1	<max string>	This variable specifies a sequence of commands that are executed when a breakpoint halts program execution. See the b command.
regstyle	sw	[hw sw]	This variable defines whether hardware or software names are displayed in the l command. See the l command.
showsym	yes	[no yes]	Show or don't show symbols when doing trace and disassembly.
validpc	_ftext etext	<low high>	This variable specifies the range of valid PC values during program tracing. See the trace command.
regsize	32	[32 64]	Register display size. Only valid on architectures with 64 bit registers.

Misc. variables

The following variables are not actually used by PMON itself but are set up accordingly and passed to a downloaded program in the environment when started:

Variable	Description
busclock	Normally set to the clock speed of the processors external bus.
cpuclock	Normally set to the internal pipeline clock speed of the processor.
memsize	Set to the number of megabytes of memory.
systype	Set to a unique name that identifies the target platform model.
l3cache	This variable will be set to the size of the Level3 cache on systems with this feature. On other platforms it is not available.

vxWorks	This string variable maintained by PMON2000 is used by vxWorks. It is stored in non-volatile memory in a location known by the vxWorks BSP for the target platform. The string is also copied to ram in the same way as the vxWorks boot do it. See Wind River documentation for details.